Tracking Freeriders in Gossip-Based Content Dissemination Systems

Rachid Guerraoui, Kévin Huguenin, Anne-Marie Kermarrec, Maxime Monod, Swagatika Prusty, Aline Roumy

Tracking Freeriders in Gossip-Based Content Dissemination Systems[☆,☆☆]

Rachid Guerraoui[a], Kévin Huguenin[a,1,*], Anne-Marie Kermarrec[b], Maxime Monod[c,2], Swagatika Prusty[d,3], Aline Roumy[b]

[a] *EPFL, School of Computer and Communication Systems, Lausanne, Switzerland.*
[b] *INRIA Rennes – Bretagne Atlantique, Campus de Beaulieu, Rennes, France.*
[c] *NXC, Lausanne, Switzerland.*
[d] *Department of Computer Science, University of Massachusetts Amherst, Amherst, MA, USA.*

Abstract

Gossip-based protocols have proven very efficient for disseminating high-bandwidth content such as video streams in a peer-to-peer fashion. However, for the protocols to work, nodes are required to collaborate by devoting a fraction of their upload bandwidth, a scarce resource for some of them, to forward the content they receive to other nodes. Consequently, such protocols suffer from freeriding, a common phenomenon on the Internet, which consists in selfishly benefiting from the system without contributing its fair share. Due to the dynamic nature and the inherent randomness of gossip protocols and to the high scalability requirements of video streaming systems, detecting freeriders is a difficult challenge.

This paper presents LiFTinG, the first protocol for detecting freeriders, including colluding ones, in gossip-based content dissemination systems with asymmetric data exchanges. In addition, LiFTinG is still able to detect freeriders when network coding, a widely used technique to improve the efficiency of content dissemination, is used. LiFTinG relies on nodes to track abnormal behavior by cross-checking the history of their previous interactions and exploits the fact that nodes pick neighbors at random to prevent colluding nodes from mutually covering up their bad actions.

We present a methodology for setting the parameters of LiFTinG to their optimal value, based on a theoretical analysis and we quantify theoretically the performance of LiFTinG. We derive, based on simulations, the optimal strategy of freeriders by taking into account, through a utility function, the benefit of freeriding and the probability of being detected. In addition to these simulations, we report on the deployment of LiFTinG on PlanetLab. In a 300-node system, where a stream of 674 kbps is broadcasted, LiFTinG incurs a maximum overhead of only 8% and provides good detection results: For instance, with 10% of freeriders decreasing their contribution by up to 30%, LiFTinG detects 86% of the freeriders after only 30 seconds and wrongfully expels only a few honest nodes (most of them actually being buggy).

Keywords: High-bandwidth content dissemination, Peer-to-Peer (P2P), Free riding, Distributed verifications.

1. Introduction

Gossip protocols have been successfully applied to decentralize large-scale high-bandwidth content dissemination, such as video streaming [12, 14, 15]. Such systems are *asymmetric*[4]: nodes propose packet identifiers to a dynamically changing subset of random nodes. Packets can be either chunks of the file or stream, or combinations of such chunks when coding is used (e.g., random linear combinations in [9, 23, 56]). These nodes, in turn, request packets of interest, that are subsequently pushed by the proposer. In such a three-phase protocol, gossip is used to disseminate content location, whereas the content itself is explicitly requested and served, in order to avoid serving redundant content. These protocols are commonly used for high-bandwidth content dissemination with gossip, e.g., [12, 14, 15, 36] (a similar scheme is also present in mesh-based systems, e.g., [35, 39, 55, 58, 59] – see [60] for a comprehensive survey of peer-to-peer live streaming protocols).

The efficiency of such protocols highly relies on the willingness of participants to collaborate, i.e., to devote a fraction of their resources, namely their upload bandwidth, to the system. Yet, some of these participants might be tempted to *freeride* [3, 25, 34], i.e., not contribute their fair share of work,

[☆] This article is a revised and extended version of a paper that appears in the Proceedings of the ACM/IFIP/USENIX 11th International Middleware Conference (Middleware '10) [19].

[☆☆] This work was partially supported by the ERC Starting Grant GOSSPLE 204742.

[*] Corresponding author

Email addresses: rachid.guerraoui@epfl.ch (Rachid Guerraoui), kevin.huguenin@epfl.ch (Kévin Huguenin), anne-marie.kermarrec@inria.fr (Anne-Marie Kermarrec), maxime.monod@nxc.ch (Maxime Monod), aline.roumy@inria.fr (Aline Roumy)

[1] This research was partially carried out while Kévin Huguenin was working for his PhD at Université de Rennes 1 / IRISA, France.

[2] This research was partially carried out while Maxime Monod was working for his PhD at EPFL, Lausanne, Switzerland. Maxime Monod was partially funded by the Swiss National Science Foundation with grant 20021-113825.

[3] Parts of this research were carried out while Swagatika Prusty was doing an internship at EPFL, Lausanne, Switzerland.

[4] Throughout the paper, asymmetry refers to the protocol, i.e., the fact that nodes push content without expecting any content in return, not to the heterogeneity of the nodes' capabilities. See [15] for a study on this latter topic.

especially if they could still benefit from the system. Freeriding is common in large-scale systems deployed in the public domain [1] and significantly degrades the overall performance in bandwidth-demanding and delay-sensitive applications such as streaming. In addition, freeriders might collude (e.g., as evidenced in the Maze peer-to-peer sharing system [37]), i.e., collaborate to decrease their individual contribution and the contribution of the coalition and mutually cover up their misbehaviors to circumvent detection mechanisms.

Although gossip protocols are almost not affected by crashes [13, 31], high-bandwidth content dissemination with gossip clearly suffers more from freeriders than from crashes. Indeed, when content is pushed in a single phase, a freerider is equivalent to a crashed node (if TCP, or a similar flow control protocol, is not used). Both crashed nodes and freeriders consume bandwidth (as content is pushed to them) and they do not provide upload bandwidth. In three-phase protocols, however, crashed nodes do not provide upload bandwidth anymore, nor do they consume bandwidth, as they do not request content from proposers after they crash. On the contrary, freeriders decrease their contribution, yet keep requesting content.

A widely used solution to counter freeriding is to use Tit-for-Tat (TfT) incentives (inspired by the BitTorrent [10] file-sharing system): TfT-based content dissemination solutions (e.g., FlightPath [36]) make nodes contribute as much as they benefit by enforcing balanced *symmetric* exchanges. However, so-called symmetric systems do not perform as well as asymmetric systems in terms of efficiency and scalability for live streaming [5].

In practice, many proposals (e.g., [12, 35, 55, 59]) consider, instead of symmetric exchanges, asymmetric exchanges where nodes are supposed to altruistically serve content to other nodes, i.e., without asking anything in return, where the benefit of a node is not directly correlated to its contribution but rather to the global *health* of the system. The correlation between the benefit and the contribution is not immediate. However, such correlation can be artificially established, in a punitive way, by means of verification mechanisms that ensure that nodes that do not contribute their fair share do not benefit anymore from the system. Freeriders are by definition rational profit-maximizing entities. Therefore, in the presence of punitive mechanisms, they can then be defined as nodes that decrease their contribution as much as possible while keeping the probability of being expelled low.

In this work, we consider a generic three-phase gossip protocol where data is disseminated following an asymmetric push scheme. Data can be transmitted in a coded form, more specifically, random linear combinations [23]. In this context, we propose LiFTinG, a lightweight mechanism to track freeriders. To the best of our knowledge, LiFTinG is the first protocol for securing asymmetric gossip protocols (even when coding is used) against possibly colluding freeriders. At the core of LiFTinG lies a set of deterministic and randomized distributed verification procedures based on *accountability* (i.e., each node maintains a digest of its past interactions). Deterministic procedures check, by cross-checking nodes' logs, that the content received by a node is actually further propagated following the protocol (i.e., to the right number of nodes within an acceptable delay). By using statistical techniques, randomized procedures check that the interactions of a node are evenly distributed in the system. Interestingly enough, the strong randomness and the high dynamics of gossip protocols, which might be considered at first glance as a barrier to properly monitor nodes, happens to help in tracking freeriders. Indeed, LiFTinG exploits the very fact that nodes pick neighbors at random to prevent collusion: As a node interacts with a large subset of the nodes chosen at random, this drastically limits its opportunity to freeride without being detected, because this prevents it from deterministically choosing colluding partners that would cover it up.

Designing such a system raises a number of challenges, including scalability, bandwidth usage, and performance of detection in the presence of message losses and untruthful reports from nodes. LiFTinG is scalable and lightweight as it relies neither on a (trusted) central authority (e.g., PKI, reputation server) nor on heavyweight cryptography and incurs only very low overhead in terms of bandwidth. In addition, LiFTinG is fully decentralized as nodes are in charge of verifying each others' actions. Finally, LiFTinG provides a good probability of detecting freeriders and keeps low the probability of false positives, i.e., inaccurately classifying a correct node as a freerider, by using mechanisms which, based on the results of our analytical analysis, (i) de-incentivize nodes from reporting wrongful accusations against other nodes and (ii) compensate the effect of message losses.

We give analytical results backed up with simulations that provide means to set the parameters of LiFTinG in a real environment. Moreover, our theoretical results can be used as input for a game-theoretical study of the system because they provide expressions (or bounds) of the key performance metrics including the probability of detection, the false positive rate, and the expected benefit. In addition, we deployed LiFTinG over PlanetLab, where a stream of 674 kbps is broadcast to 300 PlanetLab nodes with their upload bandwidth capped at 1,000 kbps for increased realism, and we report on LiFTinG's performance in practice. In order to illustrate the importance of countering freeriders and the performance of LiFTinG, consider the following high-level experimental results: In the presence of freeriders, the health of the system (i.e., the proportion of nodes able to receive the stream in function of the stream lag, i.e., cumulative distribution function) degrades significantly, compared to a system where all nodes follow the protocol. Figure 1 shows a clear drop between the plain line (no freeriders) and the dashed line (25% of freeriders). With LiFTinG, and assuming that freeriders keep their probability of being expelled lower than 50%, the performance is close to the baseline.

In this setting, LiFTinG incurs a maximum network overhead of only 8%. When freeriders decrease their contribution by 30%, LiFTinG detects 86% of the freeriders and wrongly expels 12% of honest nodes, after only 30 seconds. Most of the wrongly expelled nodes deserve it, in a sense, as their actual contribution is smaller than required. However, this is due to poor capabilities, as opposed to freeriders that deliberately decrease their contribution.

The rest of the paper is organized as follows. Section 2

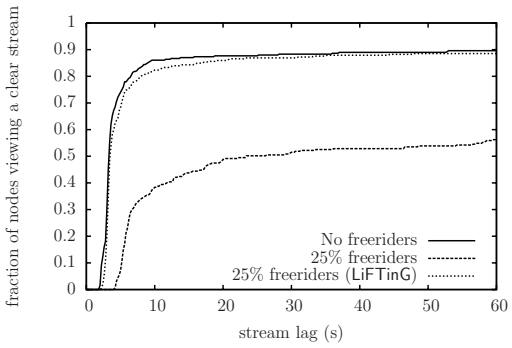

Figure 1: System efficiency in the presence of freeriders.

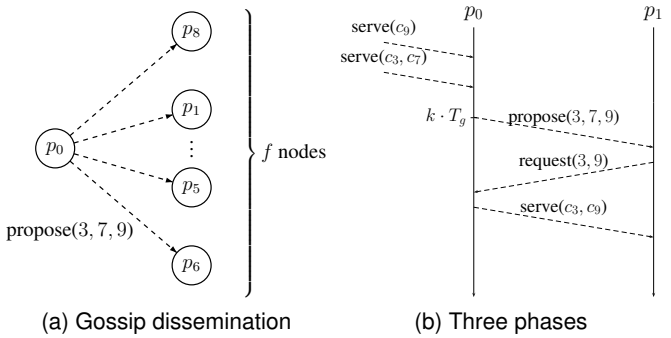

Figure 2: Three-phase generic gossip.

describes our illustrative gossip protocol and Section 3 lists and classifies the opportunities for nodes to freeride in such a content-dissemination protocol. Section 4 presents LiFTinG and Section 5 formally analyzes its performance backed up by extensive simulations. Section 6 reports on the deployment of LiFTinG over the PlanetLab testbed. Section 7 reviews related work. Section 8 concludes the paper.

2. Model and Gossip Protocol

We consider a system of n nodes that communicate over lossy links (e.g., UDP) and that can receive incoming data from any other node in the system. More specifically, the nodes that are behind a NAT or a firewall make use of the Internet Gateway Device Protocol (through Universal Plug'n'Play) to dynamically add translation rules at the router or implement UDP NAT traversal techniques ("hole punching") such as STUN [48]. Relay-based techniques can also be used [32]). In addition, nodes can pick uniformly at random a set of nodes in the system. This is achieved by using full membership (i.e., the nodes know the list of all other nodes in the system) or a random peer sampling protocol, e.g., [27, 33]. Such sampling protocols can be made robust to byzantine attacks by using techniques such as Brahms [6]. Indeed, a node might be tempted to tamper with the peer sampling service in order to be chosen, and thus served content, more frequently by other nodes.

A source broadcasts a data stream to all nodes by using a three-phase gossip protocol (e.g., [12, 14]). The content is split into multiple chunks uniquely identified by IDs. In short, each node periodically proposes a set of chunks it receives to a set of random nodes. Upon reception of a proposal, a node requests the chunks it needs–essentially those it does not have already– and the sender then serves them. All messages are sent over UDP. The three phases are illustrated in Figure 2b.

Proposal phase. A node periodically, i.e., at every gossip period T_g, picks uniformly at random a set of f nodes and proposes to them (as depicted in Figure 2a) the set $\mathcal{P}$ of chunks it has received since its last propose phase. The size f of the node set, namely the *fan-out*, is the same for all nodes and kept constant over time (the fan-out is typically set to a value slightly larger than $\ln(n)$ [31], that is $f = 12$ for a $10,000$-node system). Such a gossip protocol follows an *infect-and-die* process, as once a node proposes a chunk to a set of nodes, it does not propose it anymore.

Request phase. Upon reception of a proposal of a set $\mathcal{P}$ of chunks, a node determines the subset of chunks $\mathcal{R}$ it needs and requests these chunks.

Serving phase. When a proposing node receives a request corresponding to a proposal, it serves the chunks requested. If a request does not correspond to a proposal, it is ignored. Similarly, nodes only serve chunks that are effectively proposed, i.e., chunks in $\mathcal{P} \cap \mathcal{R}$.

Network coding. To increase the efficiency of the dissemination, coding techniques can be used: by proposing *combinations* of chunks instead of proposing chunks, the probability of proposing, and thus of pushing, useful content increases. For instance, random linear network coding [23] was successfully used in Avalanche [18], in SPANC [9], and in R2 [56]. When random linear network coding is used, nodes propose *linear combinations* (i.e., bitwise XORs) of the chunks they receive during the last period. A node proposes one linear combination for each chunk it received over the last gossip period to each of the f nodes it contacts. The coefficients of the combination are picked at random from a Galois field GF(2^q). To propose a linear combination of chunks, a node sends a set of pairs (ID,coefficient) instead of a single ID. For instance, to propose the combination $2 \cdot c_3 \oplus c_5$, a nodes sends $\{(3, 2), (5, 1)\}$. To ensure an optimal utility of the proposed combinations, the proposer makes sure they are linearly independent. Checking linear independence and decoding the original data chunks of the stream are achieved through Gauss elimination. For the receiver, the packets of interest are those that are linearly independent with the packets it has received so far. The receiver requests only such packets from the proposer.

3. The Freeriding Problem

Nodes are either honest or freeriders; we denote by m the number of freeriders. Honest nodes strictly follow the protocol, including the verification procedures specified in LiFTinG. Freeriders, however, allow themselves to deviate from the protocol in order to minimize their contribution while maximizing their utility. In addition, freeriders can adopt any behavior in order to not be expelled, including lying to verifications,

or covering up colluding freeriders' bad actions. More generally, freeriders are rational entities: They behave in such a way that their utility is maximized; their utility being a decreasing function of their upload bandwidth usage and an increasing function of the quality of the stream they receive. Note that in our model, we assume that freeriders do not wrongfully accuse (honest) nodes (In [7], the authors propose techniques to deal with such accusations). This is motivated by the fact that causing honest nodes to be expelled *(i)* does not increase the benefit of freeriders, *(ii)* does not prevent them from being detected, i.e., detection is based solely on the suspected node's behavior regardless of other nodes' behaviors (details in Section 5.1), and finally *(iii)* leads to an increased proportion of freeriders, degrading the benefit of all nodes (including freeriders). This phenomenon is known as the *tragedy of the commons* [21]. Note that there are still some advantages for freeriders to wrongfully accuse other nodes, e.g., discredit the entire reputation system. Existing solutions can be used to mitigate the effect of such dishonest behaviors. For instance, it can be enforced that nodes accuse only the nodes they interact with. In addition, one can limit the number of nodes a node can accuse (per minute); this would have a limited effect on legitimate accusations provided that the proportion of freeriders remains low. Other existing solutions (e.g., [40], see [24] for a comprehensive survey) propose to modulate accusations by the credibility/trustworthiness (which can be the reputation of the node itself) of the accusing node. More complex solutions can be used; for instance in SumUp [54], the nodes propagate their accusations (i.e., votes in the original version) along the edges of a social network in order to mitigate the effect of a coalition of related nodes that try to jointly wrongfully accuse other nodes. Such solutions however, often require some infrastructure and extra information which do not match the requirements of LiFTinG in terms of decentralization, scalability and reactivity.

Freeriders deviate from the gossip protocol in the following ways: *(i)* decrease the number of partners to communicate with, *(ii)* bias the partner selection, *(iii)* drop messages they are supposed to send, or *(iv)* modify the content of the messages they send. In the following section, we provide an exhaustive list of all possible attacks in each phase of the protocol, we discuss their motivations and effects, and then we extract and classify those that can increase the individual utility of a freerider or the common utility of colluding freeriders. Throughout the paper, the attacks that require collusion between some nodes or profit to colluding nodes are denoted with a '$\star$'.

Proposal phase. During the first phase, a freerider can *(i)* communicate with less than f nodes, *(ii)* propose less chunks than it should (less linear combinations, or linear combinations of less received packets when network coding is used), *(iii)* select as communication partners only a specific subset of nodes, and *(iv)* reduce its proposing rate.

(i) **Decreasing fan-out** By proposing chunks to $\hat{f} < f$ nodes per gossip period, the freerider trivially reduces the potential number of requests and thus the probability of serving chunks. Therefore, its contribution in terms of the amount of data uploaded is decreased and its utility increases.

(ii) **Invalid proposal** A proposal is valid if it contains every chunk received in the last gossip period. Proposing only a subset of the chunks received in the last period obviously decreases the number of requested chunks. However, a freerider has no interest in proposing chunks it does not have as, unlike with TfT-based protocols, uploading chunks to a node does not imply that it sends chunks in return. In other words, proposing more (and possibly fake) chunks does not increase the benefit of a node and thus does not need to be considered.

(iii) **Biasing the partners selection ($\star$)** Considering a group of colluding nodes, a freerider might want to bias the random selection of nodes to favor its colluding partners, so that the group's benefit increases.

(iv) **Increasing the gossip period** A freerider can increase its gossip period, leading to less frequent proposals advertising more, but "older", chunks per proposal. This implies a decreased interest of the requesting nodes and thus a decreased contribution for the sender. This is due to the fact that an old chunk has a lower probability of being of interest as it becomes more replicated over time.

Pull-request phase. Nodes are expected to request only chunks that have been proposed to them. A freerider would increase its benefit by opportunistically requesting extra chunks (even from nodes that did not propose these chunks). The dissemination protocol itself prevents this misbehavior by automatically dropping such requests.

Serving phase. In the serving phase, freeriders can *(i)* send only a subset of what was requested or *(ii)* send junk. The first obviously decreases the freeriders' contribution, as they serve fewer chunks than they are supposed to. However, as we mentioned above, in the considered asymmetric protocol, a freerider has no interest in sending junk data, because it does not receive anything in return for what it sends.

Summary. Analyzing the basic gossip protocol in detail enables us to identify the possible attacks. Interestingly enough, these attacks share similar aspects and can thus be gathered into three classes that dictate the rationale along which our verification procedures are designed.

The first is *quantitative correctness* that characterizes the fact that a node effectively proposes to the correct number of nodes (f) at the correct rate ($1/T_g$). Assuming this first aspect is verified, two more aspects must be further considered: *causality* that reflects the correctness of the deterministic part of the protocol, i.e., received chunks must be proposed in the next gossip period (as depicted in Figure 2b); and *statistical validity* that evaluates the fairness (with respect to the distribution specified by the protocol) in the random selection of communication partners.

4. Tracking Freeriders in Gossip

To address the problem of freeriders in epidemic protocols, we propose LiFTinG, a lightweight protocol for freerider tracking in gossip, that encourages nodes, in a dissuasive way, to

contribute their fair share to the system, by means of distributed verifications. LiFTinG consists of *(i) direct* verifications and *(ii) a posteriori* verifications. Verifications, that require more information than what is available at the verifying node and the inspected node, are referred to as *cross-checking*. This essentially consists in several nodes grouping together their information to effectively detect misbehaviors (that were committed by the inspected node) that could not be detected solely based on the information they hold individually. Cross-checking requires nodes to communicate and therefore incurs communication overhead. In order to control the overhead of LiFTinG, the frequency at which such verifications are triggered is controlled by a parameter $p_{cc} \in [0, 1]$, as described in Section 4.2. Verifications can either lead to the emission of *blames* or directly to *expulsion*, depending on the gravity of the misbehavior.

Direct verifications are performed regularly while the protocol is running: the nodes' actions are directly checked. Direct verifications aim at checking that all chunks requested are served and that all chunks served are further proposed to a correct number of nodes, i.e, they check the *quantitative correctness* and *causality*. Direct verifications are composed of *(i)* direct checking and *(ii)* direct cross-checking.

A posteriori verifications are run sporadically. They require each node to maintain a log of its past interactions, namely a *history*. In practice, a node stores a digest of the events that occurred in the last h seconds (i.e., a sliding window), corresponding to the last $n_h = h/T_g$ gossip periods. The history is audited to check the statistical validity of the random choices made when selecting communication partners. In LiFTinG, an *entropic check* is used as described in Section 4.4. The veracity of the history is verified by cross-checking the involved nodes, namely a posteriori cross-checking.

We present the blaming architecture in Section 4.1 and present direct verifications in Section 4.2. As freeriders can collude to not be detected, we expose how they can cover up each other's misbehaviors in Section 4.3 and address this breach in Section 4.4. The different attacks and corresponding verifications are summarized in Table 1.

Attack	**Type**	**Detection**
fan-out decrease ($\hat{f} < f$)	quantitative	direct cross-check
partial propose ($\mathcal{P}$)	causality	direct cross-check
partial serve ($\lvert\mathcal{S}\rvert < \lvert\mathcal{R}\rvert$)	quantitative	direct check
bias partners selection ($\star$)	entropy	entropic check *a posteriori* cross-check

Table 1: Summary of attacks and associated verifications.

4.1. System Architecture

In LiFTinG, the detection of freeriders is achieved by means of a score assigned to each node. When a node detects that some other node is freeriding, it emits a blame message containing a *blame value* (i.e., essentially a real number) against the suspected node. Summing up the blame values of a node results in a score. For scores to be meaningful, blames emitted by different verifications should be comparable and homogeneous.

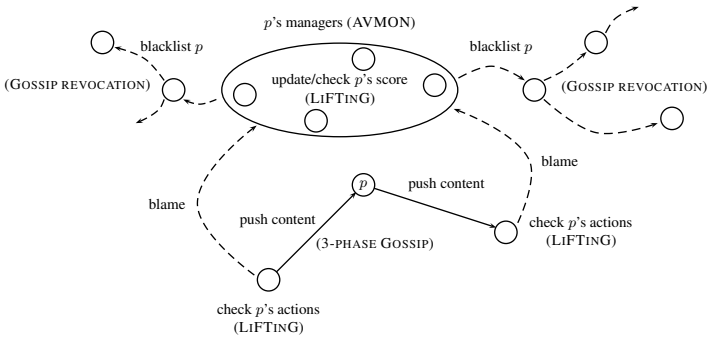

Figure 3: Overview of LiFTinG's architecture and functionning.

In order to collect blames targeted at a given node and to maintain its score, each node is monitored by a set of other nodes, named *managers*, distributed among the participants. Blame messages about a node are sent to its managers. When a manager detects that the score of a node p it monitors drops beyond a *fixed* threshold (the design choice of using a fixed threshold is explained in Section 5.1), it spreads – through gossip – a revocation message against p, thus making the nodes of the system progressively remove p from their membership. A general overview of the architecture of LiFTinG is given in Figure 3.

The blaming architecture of LiFTinG is built on top of the AVMON [43] monitoring overlay[5]. In AVMON, nodes are assigned a *fixed-size* set of M *random* managers *consistent* over time, which makes it very appealing in our setting, specifically a dynamic peer-to-peer environment subject to churn with possibly colluding nodes. The fact that the number M of managers is constant makes the protocol scalable, as the monitoring load at each node is independent of the system size. Randomness prevents colluding freeriders from covering each other up, and consistency makes long-term blame history at the managers, and thus long-term follow up, possible. The monitoring relationship is based on a hash function and can be advertised in a gossip-fashion by piggybacking node's monitors in the view maintenance messages (e.g., exchanges of local views in the distributed peer-sampling service). Doing so, nodes quickly discover other nodes' managers – and are therefore able to blame the nodes if necessary – even in the presence of churn. In addition, nodes can locally verify (i.e., without the need for extra communication) whether the node-to-managers mapping is correct by hashing the nodes' IP addresses, thus preventing freeriders from forging fake or colluding managers. If a manager does not map correctly to a node, a revocation against the concerned node is sent.

4.2. Direct Verifications

In LiFTinG, two direct verifications are used. The first aims to ensure that every requested chunk is served. It is called *direct check*. As detection can be done locally and thus does not incur any bandwidth overhead, it is always performed. If

[5]Note that other monitoring systems, such as PeerMint which makes use of a Distributed Hash Table (DHT) to store and maintain node profiles or AlliaTrust [17], could be used.

some requested chunks are missing, the requesting node blames the proposing node by $f/|\mathcal{R}|$ (where $\mathcal{R}$ is the set of requested chunks) for each chunk that has not been delivered.

The second verification checks that received chunks are further proposed to f nodes within the next gossip period. This is achieved by a *cross-checking* procedure that works as follows: a node p_1 that received a chunk c_i from p_0 during the previous gossip period acknowledges to p_0 that it proposes c_i to a set of f nodes. Then, p_0 sends confirm requests (with probability p_{cc}) to the set of f nodes to check whether they effectively received a propose message from p_1 containing c_i. The f witnesses reply to p_0 with answer messages confirming whether p_1's acknowledgment to p_0.

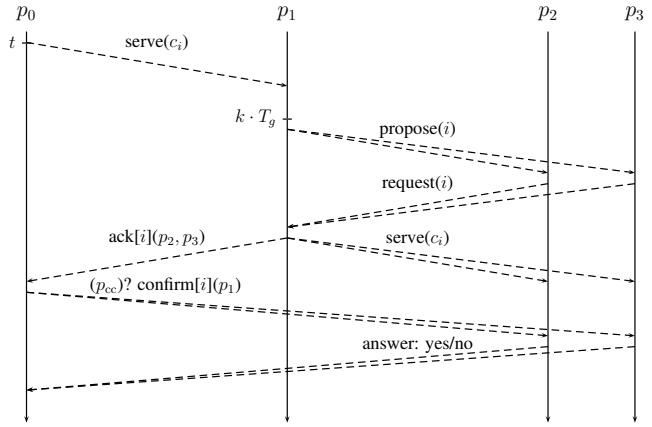

Figure 4: Cross-checking protocol.

Figure 4 depicts the message sequence that composes a direct cross-checking verification (with a fan-out of 2 for the sake of readability). The blaming mechanism works as follows: *(i)* if the ack message is not received, the verifier p_0 blames the verified node p_1 by f, and *(ii)* for each missing or negative answer message, p_0 blames p_1 by 1. In the case where network coding is used, the linear combinations a node sends (more specifically, the coefficients) are specified by the nodes that served it during the last gossip period, in order to prevent the proposing node from misbehaving (e.g., carefully choosing and proposing combinations that are not of interest to the receiver). For instance, when p_0 serves p_1, it specifies a set of coefficients for each packet it serves, e.g., $\{3, 4, \dots\}$. During the next gossip period, p_1 must propose the following combination: 3 times the first packet it received during this gossip period, plus 4 times the second one (if any), and so on and so forth. When p_0 asks for a confirmation from p_2 and p_3, it includes the list of packets it served to p_1 and the coefficients it specified to p_1. Then, p_2 and p_3 can verify if the combinations proposed by p_1 match the specifications of p_0.

As the verification messages (i.e., ack, confirm and confirm responses) for the direct cross-checking are small and in order to limit the subsequent overhead of LiFTinG, direct cross-checking is done exclusively with UDP. The blames corresponding to the different attacks are summarized in Table 2.

Blames emitted by the direct verification procedures of LiFTinG are summed into a score reflecting the nodes' behaviors. For this reason, blame values must be comparable and homogeneous. This means that two misbehaviors that reduce a freerider's contribution by the same amount should lead to the same value of blame, regardless of the misbehaviors and the verification.

Attacks	**Blame values**
fan-out decrease ($\hat{f} < f$)	$f - \hat{f}$ from each verifier
partial propose	1 (per invalid proposal) from each verifier
partial serve ($\lvert\mathcal{S}\rvert < \lvert\mathcal{R}\rvert$)	$f \cdot (\lvert\mathcal{R}\rvert - \lvert\mathcal{S}\rvert)/\lvert\mathcal{R}\rvert$ from each requester

Table 2: Summary of attacks and associated blame values.

We consider a freerider p_f that received c chunks and wants to reduce its contribution by a factor δ ($0 \leq \delta \leq 1$). To achieve this goal, p_f can do one of the following: *(i)* propose the c received chunks to only $\hat{f} = (1-\delta) \cdot f$ nodes, *(ii)* propose only a proportion $(1-\delta)$ of the chunks it received, or *(iii)* serve only $(1-\delta) \cdot |\mathcal{R}|$ of the $|\mathcal{R}|$ chunks it was requested. For the sake of simplicity, we assume that $\hat{f}, c \cdot \delta, c/f$ and $\delta \cdot |R|$ are all integers. The number of verifiers, that is, the number of nodes that served the c chunks to p_f is called the *fan-in* (f_{in}). On average, we have $f_{in} \simeq f$ and each node serves c/f chunks [16].

We now derive, for each of the three aforementioned misbehaviors, the blame value emitted by the direct verifications.

(i) Fan-out decrease (direct cross-check): If p_f proposes all the c chunks to only $\hat{f}$ nodes, it is blamed by 1 by each of the f_{in} verifiers, for each of the $f - \hat{f}$ missing "propose target". This results in a blame value of $f_{in} \cdot (f - \hat{f}) = f_{in} \cdot \delta \cdot f \simeq \delta f^2$.

(ii) Partial propose (direct cross-check): If p_f proposes only $(1-\delta) \cdot c$ chunks to f nodes, it is blamed by f by each of the nodes that provided at least one of the missing chunks. A freerider therefore has interest in removing from its proposal those chunks originating from the smallest subset of nodes. In this case, its proposal is invalid from the standpoint of $\delta \cdot f_{in}$ verifiers. This results in a blame value of $\delta \cdot f_{in} \cdot f \simeq \delta \cdot f^2$.

(iii) Partial serve (direct check): If p_f serves only $(1-\delta) \cdot |\mathcal{R}|$ chunks, it is blamed by $f/|\mathcal{R}|$ for each of the $\delta \cdot |\mathcal{R}|$ missing chunks by each of the f requesting nodes. This again results in a blame value of $f \cdot (f/|\mathcal{R}|) \cdot \delta \cdot |\mathcal{R}| = \delta \cdot f^2$.

The blame values emitted by the different direct verifications are therefore homogeneous and comparable on average, because all misbehaviors lead to the same amount of blame for a given degree of freeriding δ. Thus, they result in a consistent and meaningful score when summed up.

4.3. Fooling the Direct Cross-Check (⋆)

When a set of freeriders collude, they lie to verifications to mutually cover up their misbehaviors. Consider the situation depicted in Figure 5a, where p_1 is a freerider. If p_0 colludes with p_1, then it will not blame p_1, regardless of p_2's answer.

Similarly, if p_2 colludes with p_1, then it will answer to p_0 that p_1 sent a valid proposal, regardless of what p_1 sent. Even when neither p_0 nor p_2 collude with p_1, p_1 can still fool the direct cross-checking – thanks to a colluding third party by implementing a *man-in-the-middle attack* as depicted in Figure 5b. Indeed, if a node p_7 colludes with p_1, then p_1 can tell p_0 it sent a proposal to p_7 and tell p_2 that the chunk originated from p_7. Doing this, both p_0 and p_2 will not detect that p_1 sent an invalid proposal. The *a posteriori verifications* presented in the next section address this issue.

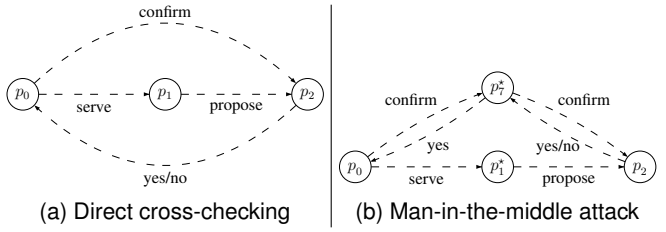

Figure 5: Direct cross-checking and attack. Colluding nodes are denoted with a '$\star$'.

4.4. A Posteriori Verifications

As stated in the analysis of the gossip protocol, the random choices made in the partners selection must be checked. In addition, the example described in the previous section, where freeriders collude to circumvent direct cross-checking, highlights the need for statistical verification of the nodes' past communication partners.

The history of a node that biased its partner selection contains a relatively large proportion of colluding nodes. If only a small fraction of colluding nodes is present in the system, they will appear more frequently than honest nodes in each other's histories and can therefore be detected. Technically speaking, the IDs of the nodes a node communicates with are a sequence of realizations of independent identically distributed (i.i.d.) random variables drawn from a uniform distribution (across the whole set of nodes in the system). Determining if an observed sequence of node ids is drawn from a given distribution can be achieved through a statistical *goodness-of-fit* test [52]. Below, we present three variants of such tests.

Statistical verifications operate as follow (see Figure 6): once in a while, each node picks a random node (e.g., one of the nodes it manages) and verifies its local history over the last h seconds. When inspecting the history of p, the verifier computes the number of occurrences of each node in the set of proposals sent by p during the last h seconds. We denote by $\mathcal{F}_h$ as the multiset of nodes to whom p sent a proposal during this period (a node could indeed appear more than once in $\mathcal{F}_h$) . The distribution $\tilde{d}_h$ of nodes in $\mathcal{F}_h$ characterizes the randomness of the partners selection. We denote by $\tilde{d}_{h,i}$ the number of occurrences of node i ($i \in \{1, \ldots, n\}$) in $\mathcal{F}_h$ normalized by the size of $\mathcal{F}_h$. Then, a statistical test is run on the observed distribution.

The Kullback-Leibler divergence. Assessing the similarity of two distributions, i.e., the distribution $\tilde{d}$ of p_1's history and the uniform distribution, can be achieved with the Kullback-Leibler divergence [11]. When the reference distribution is the

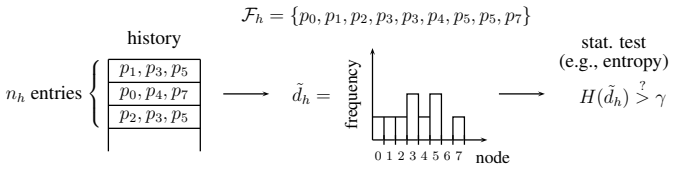

Figure 6: Entropic check on proposals ($f = 3$).

uniform distribution, this comes down to computing the Shannon entropy of the observed distribution and to comparing the value obtained to a threshold γ ($0 \leq \gamma \leq \log_2(n_h f)$).

$$H(\tilde{d}_h) = -\sum_{i=1}^{n_h f} \tilde{d}_{h,i} \log_2(\tilde{d}_{h,i}) \quad (1)$$

The entropy is maximum when every node of the system appears at most once in $\mathcal{F}_h$ (assuming $n > |\mathcal{F}_h| = n_h f$). In this case, it is equal to $\log_2(n_h f)$. As the peer selection service might not be perfect, the threshold γ must be tolerant to a small deviation, with respect to the uniform distribution to avoid *false positives* (i.e., honest nodes being blamed). In fact, entropic and statistical tests similar to those presented in this section are often used to assess the quality of random peer sampling algorithms.

The χ^2 and the Kolmogorov-Smirnov test. These tests evaluate the likelihood that an observed sample is drawn from a specific distribution (here the uniform distribution). This is achieved by computing a function, namely a statistic, of the observed sample. Under the hypothesis that the observed sample is indeed drawn from the specific distribution, the statistics follows a well-known distribution. For instance, for the χ^2 test for assessing the goodness of fit of the uniform distribution, the statistic is $F_{\chi^2}(\tilde{d}_h) = \sum_{i=1}^{n_h f} (\tilde{d}_{h,i} - 1/n)^2$ and F_{χ^2} follows a χ^2 distribution with $n_h f - 1$ degrees of freedom. The likelihood of the hypothesis is then evaluated by using the statistics table of the χ^2 distribution. For the Kolmogorov-Smirnov test, the statistics (see [52]) follows a Kolmogorov distribution.

For the sake of simplicity, LiFTinG makes use of the entropic check. Details on how to dimension the threshold γ are given in Section 5.2.

The statistical check must be coupled with an *a posteriori* cross-checking verification procedure to guarantee the validity of the inspected node's history. Cross-checking is achieved by polling all or a subset of the nodes mentioned in the history for an acknowledgment. The inspected node is blamed by 1 for each proposal in its history that is not acknowledged by the alleged receiver. Hence, an inspected freerider replacing colluding nodes by honest nodes in its history in order to pass the entropic check will not be covered by the honest nodes and will thus be blamed accordingly.

Because of the man-in-the middle attack presented in Section 4.2, a complementary entropic check is performed on the multi-set of nodes $\mathcal{F}'_h$ that asked the nodes in $\mathcal{F}_h$ for a confirmation, i.e., direct cross-checking. On the one hand, for an honest node p_0, $\mathcal{F}'_h$ is composed of the nodes that sent chunks to p_0 – namely its *fan-in*. On the other hand, for a freerider $p_0^\star$

that implemented the man-in-the-middle attack, the set $\mathcal{F}'_h$ of $p_0^\star$ contains a large proportion of colluding nodes – the nodes that covered it up for the direct cross-checking – and thus fails the entropic check. If the history of the inspected node does not pass the entropic checks (i.e, fan-in and fan-out), the node is expelled from the system.

Local-history auditing verifications are sporadically performed by the nodes using TCP connections. The reasons for using TCP are that *(i)* the overhead of establishing a connection is amortized because local history auditing happens sporadically and implies transferring a large amount of data, i.e., proportional to h, and that *(ii)* local auditing is very sensitive to message losses as the potential blame is much larger than for direct verifications and it can lead to expulsion from the system.

5. Parametrizing LiFTinG

This section provides a methodology to set LiFTinG's parameters. With this aim, the performance of LiFTinG, with respect to detection, is analyzed theoretically. Closed form expressions of the detection and false positive probabilities as functions of the system parameters are given. Theoretical results allow the system designer to set the system parameters, e.g., detection thresholds. The notations used throughout the section are summarized in Table 3.

This section is split into three parts. First, the design of the score-based detection mechanism is presented and analyzed by taking into account message losses. Second, the entropy-based detection mechanism is analyzed by taking into account the underlying peer-sampling service. Both depend on the degree of freeriding and on the favoring factor, i.e., how freeriders favor colluding partners. Third, the message complexity of LiFTinG is analyzed, as a function of the various system parameters, as it constitutes an important factor when choosing the values of the parameters.

Notations	Descriptions
n,m	number of nodes / freeriders
$\lvert\mathcal{R}\rvert$	number of chunks requested
f	fan-out
n_h	size of history
$\mathcal{F}_h, \mathcal{F}'_h$	multi-set of fan-out / fan-in in history
p_{dcc}	probability to trigger direct cross-checking
p_l	probability of message loss ($p_r = 1 - p_l$)
$\tilde{b}$	average value of wrongful blames
$\sigma(b)$	standard deviation of wrongful blames
r	number of gossip periods spent in the system
s	normalized score
Δ	degree of freeriding (3-uple)
$\tilde{b}(\Delta)$	average value of blames (freeriders)
$\sigma(b'(\delta))$	standard deviation of blames (freeriders)
η	detection threshold (blame-based detection)
α	probability of detection (blame-based detection)
β	probability of false positive (blame-based detection)
γ	detection threshold (entropy-based detection)

Table 3: Summary of principal notations.

5.1. Score-Based Detection

Because of message losses, all nodes can be wrongfully blamed, i.e., blamed even though they follow the protocol. In addition, freeriders are blamed for their misbehaviors. Therefore, the score distribution among the nodes is expected to be a mixture of two components corresponding respectively to those of honest nodes and freeriders. In this setting, likelihood maximization algorithms are traditionally used to decide whether a node is a freerider. Such algorithms are based on the relative score of the nodes and are thus not sensitive to wrongful blames. Effectively, wrongful blames have the same effect on honest nodes and freeriders.

However, in the presence of freeriders, two problems arise when using relative score-based detection: *(i)* freeriders are able to decrease the probability of being detected by wrongfully blaming honest nodes, and *(ii)* the score of a node joining the system is not comparable to those of the nodes already in the system. For these reasons, in LiFTinG, the effect of wrongful blames, due to message losses, is automatically compensated, and detection thus consists in comparing the nodes' compensated scores to a fixed threshold η. In short, when the compensated score of a node drops below η, the managers of that node broadcast a revocation message, thus expelling the node from the system, by using gossip.

Considering message losses independently drawn from a Bernoulli distribution of parameter p_l (we denote by $p_r = 1 - p_l$ the probability of reception), we derive a closed-form expression for the expected value of the blames applied to honest nodes by direct verifications during a given timespan. Periodically increasing all scores accordingly (i.e., by a value corresponding to the expected wrongful blames applied to the nodes because of message loss) leads to an average score of 0 for honest nodes. This way, the fixed threshold η can be used to distinguish between honest nodes and freeriders. The value of p_r used to compensate the wrongful blames in LiFTinG is the same for all the nodes; it is evaluated once for all–or at least at a low frequency (e.g., every month)–and hard-coded in the protocol, independently from LiFTinG (e.g., by experimentally evaluating the average message loss rate between trusted nodes).

5.1.1. Wrongful Blames

We now compute the expected value of the wrongful blames applied to honest nodes by direct verifications. To this end, we analyze, for each verification, the situations where message losses can cause wrongful blames, and we evaluate their average impact. For the sake of the analysis, we assume that *(i)* a node receives at least one chunk during every gossip period (and therefore it will send proposals during the next gossip period), and *(ii)* each node requests a constant number $|\mathcal{R}|$ of chunks for each proposal it receives. We consider the case where cross-checking is always performed, i.e., $p_{\text{cc}} = 1$.

Direct check (dc). For each requested chunk that has not been served, the node is blamed by $f/|\mathcal{R}|$. If the proposal is received but the request is lost (i.e., $p_r(1 - p_r)$), the node is blamed by f ((a) in Equation 2). Otherwise, when both the proposal and the request message are received (i.e., p_r^2), the node is blamed

by $f/|R|$ for each of the chunks lost (i.e., $(1-p_r)|R|$) ((b) in Equation 2). The expected blame, applied to an honest node (by its f partners), during one gossip period, due to message losses is therefore

$$\begin{aligned}\tilde{b}_{\text{dc}} &= f \cdot \left[\overbrace{p_r(1-p_r)\cdot f}^{\text{(a)}} + \overbrace{p_r^2\cdot(1-p_r)\,|\mathcal{R}|\cdot\frac{f}{|\mathcal{R}|}}^{\text{(b)}}\right] \\ &= p_r(1-p_r^2)\cdot f^2 \end{aligned} \quad (2)$$

Direct cross-checking (dcc). On average, a node receives f proposals during each gossip period. Therefore a node is subject to f direct cross-checking verifications and each verifier asks for a confirmation from the f partners of the inspected node. Let p_1 be the inspected node and p_0 a verifier. First, note that p_0 verifies p_1 only if it served chunks to p_1, which requires that its proposal and the associated request have been received (i.e., p_r^2). If at least one chunk served by p_0 or the ack has been lost (i.e., $1-p_r^{|\mathcal{R}|+1}$), p_0 will blame p_1 by f regardless of what happens next, because all the f proposals sent by p_1 are invalid from the standpoint of p_0 ((a) in Equation 3). Otherwise, that is, if all the chunks served and the ack have been received (i.e., $p_r^{|\mathcal{R}|+1}$), p_0 blames p_1 by 1 for each negative or missing answer from the f partners of p_1. This situation occurs when the proposal sent by p_1 to a partner, the confirm message or the answer is lost (i.e., $1-p_r^3$) ((b) in Equation 3).

$$\begin{aligned}\tilde{b}_{\text{cc}} &= f\cdot p_r^2\left[\overbrace{(1-p_r^{|\mathcal{R}|+1})\cdot f}^{\text{(a)}} + \overbrace{f\cdot p_r^{|\mathcal{R}|+1}(1-p_r^3)}^{\text{(b)}}\right] \\ &= p_r^2(1-p_r^{|\mathcal{R}|+4})\cdot f^2 \end{aligned} \quad (3)$$

From the previous analysis, we obtain a closed-form expression for the expected value of the blame b applied to an honest node by direct verifications due to message losses:

$$\tilde{b} = \tilde{b}_{\text{dc}} + \tilde{b}_{\text{cc}} = p_r(1+p_r-p_r^2-p_r^{|\mathcal{R}|+5})\cdot f^2 \ . \quad (4)$$

5.1.2. Freeriding Blames

The blame value b', applied to a freerider by direct verifications, depends on its *degree of freeriding* Δ that characterizes its deviation from the protocol. Formally, we define the degree of freeriding as a 3-uple $\Delta = (\delta_1, \delta_2, \delta_3)$, $0 \le \delta_1, \delta_2, \delta_3 \le 1$, so that a freerider contacts only $(1-\delta_1)\cdot f$ nodes per gossip period, proposes the chunks received from a proportion $(1-\delta_2)$ of the nodes that served it in the previous gossip period, and serves $(1-\delta_3)\cdot|\mathcal{R}|$ chunks to each requesting node. With the same methodology as for $\tilde{b}$, we get:

$$\begin{aligned}\tilde{b}'(\Delta) &= (1-\delta_1)\cdot p_r\left(1-p_r^2(1-\delta_3)\right)\cdot f^2 + \delta_2\cdot f^2 + \\ &\quad (1-\delta_2)\cdot p_r^2\cdot\left[p_r^{|\mathcal{R}|+1}(1-p_r^3(1-\delta_1)) + \right. \\ &\quad \left. (1-p_r^{|\mathcal{R}|+1})\right]\cdot f^2 \end{aligned} \quad (5)$$

Note that the gain in terms of the upload bandwidth saved by a freerider is $1-(1-\delta_1)(1-\delta_2)(1-\delta_3)$. Following the same line of reasoning, a closed-form expression of the standard deviation $\sigma(b)$ (resp. $\sigma(b'(\Delta))$ of b (resp. $b'(\Delta)$) can be derived. Note that, unlike for the computation of the expection, for the standard deviation all the sources of blame must be considered jointly as they are not independent (and the standard deviation is therefore not additive). The analysis thus needs to be more systematic than above, that is, building a binary decision tree with all the messages exchanged during a gossip period (each branch coding whether the corresponding message was received or lost) and counting the total blame for each of the cases at the leaves.

5.1.3. Scores

In order to enable the use of a fixed threshold η, the scores are compensated with respect to message losses and normalized by the number of gossip periods r the node spent in the system. At the t-th gossip period, the score of a node writes

$$s = -\frac{1}{r}\sum_{i=0}^{r}(b_{t-i} - \tilde{b}), \quad (6)$$

where b_i is the value of the blames applied to the node during the i-th gossip period.

Figure 7 depicts the distribution of compensated and normalized scores (see Formula 6) in the presence of 1,000 freeriders of degree $\delta = \delta_1 = \delta_2 = \delta_3 = 0.1$ in a 10,000-node system after $r = 50$ gossip periods. The message loss rate is set to 7%, the fan-out f to 12 and $|\mathcal{R}| = 4$. The scores of the nodes were increased by $-\tilde{b} = 72.95$, according to Formula (4). We plot separately the distribution of scores among honest nodes and freeriders. As expected, the probability density function (Figure 7a) is split into two disjoint modes separated by a gap: the lowest (i.e., left most) mode corresponds to freeriders and the highest one to honest nodes. We observe that the average score (dotted line) is close to zero (< 0.01), which means that the wrongful blames have been successfully compensated.

5.1.4. Detection

Now, we evaluate the ability of LiFTinG to detect freeriders (probability of detection α) and the proportion of honest nodes wrongfully expelled from the system (probability of false positives β). Figure 7b depicts the cumulative distribution function of scores and illustrates the notion of detection and false positives for a given value of the detection threshold.

From the previous analysis, we obtained expressions of the expectation and the standard deviation of the blames applied to honest nodes at each round due to message losses. Therefore, assuming that the b_i are independent and identically distributed (i.i.d.), we get $\mathbb{E}[s] = 0$ and $\sigma(s) = \sigma(b)/\sqrt{r}$. Using Bienaymé-Tchebychev's inequality we get:

$$\beta = P(s<\eta) \le \frac{\sigma(b)^2}{r\cdot\eta^2} \ ; \ \alpha(\Delta) \ge 1 - \frac{\sigma(b'(\Delta))^2}{r\cdot(\tilde{b'}(\Delta)-\eta)^2} \quad (7)$$

In LiFTinG, we set the detection threshold η to -9.75 so that the probability of false positive is lower than 1%; we assume that freeriders perform all possible attacks with degree δ (i.e., $\delta_1 = \delta_2 = \delta_3 = \delta$); and we observe the proportion of freeriders detected by LiFTinG for several values of δ. Figure 8 plots α as a function of δ. We observe that a node freeriding by 5%

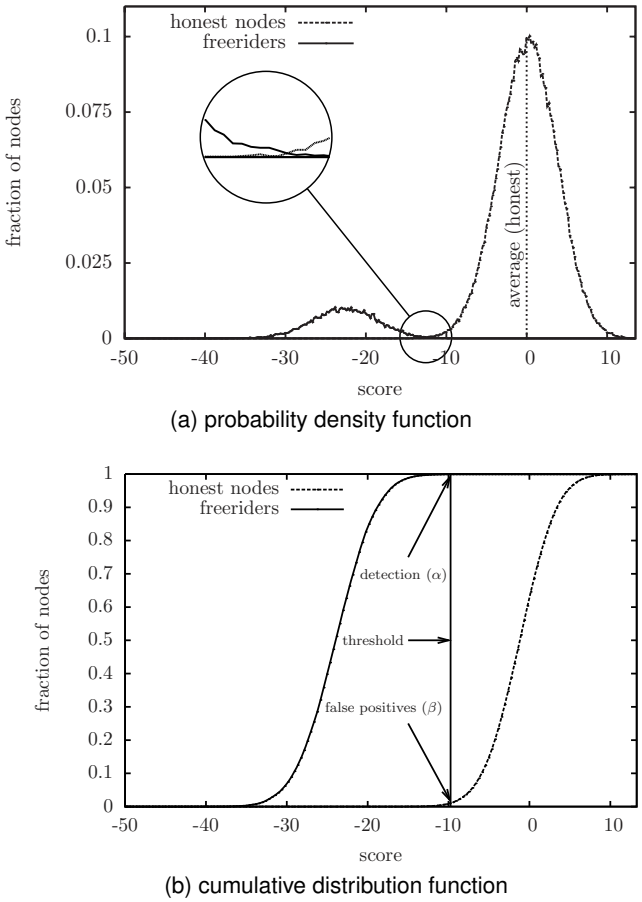

Figure 7: Distribution of normalized scores in the presence of freeriders (δ = 0.1)).

is detected with probability 0.65. Beyond 10% of freeriding, a node is detected over 99% of the time. It is commonly assumed that users are willing to use a modified version of the client application only if it increases significantly their benefits (resp. decreases their contribution). In FlightPath [36], this threshold is assumed to be around 10%. With LiFTinG, a freerider achieves a gain of 10% for $\delta = 0.035$ which corresponds to a probability of being detected of 50% (Figure 8).

5.1.5. Optimal freeriding strategy

From the previous analysis, we can extract expressions and bounds of the key factors that affect a node's utility – specifically the probability of detection (α), and the upload bandwidth savings ($1-(1-\delta_1)(1-\delta_2)(1-\delta_3)$) – as functions of the degree of freeriding. In addition, previous studies of epidemic high-bandwidth content dissemination protocols derived expressions of the global health of the system, which determines the benefits of the nodes (i.e., the quality of service), as a function of the contributions of the nodes (e.g., [5]). The coupling of LiFTinG's detection and punishment mechanisms adds a feedback loop from the nodes' behavior to their benefits. Therefore, one has all the ingredients to perform a game-theoretical study of a gossip-based dissemination protocol secured with LiFTinG.

Although performing a complete study is out of the scope of this paper, we consider a simple utility function to characterize the behavior of the freeriders when LiFTinG is used. We define the utility of a node as the benefit of seeing the stream, minus its contribution in terms of the devoted upload bandwidth. For the sake of simplicity, we assume that any node in the system (i.e., not expelled) can see the stream, and in this case the benefit is $B > 0$ (0 otherwise). For an honest node, the cost of uploading content to the other nodes is denoted by $C > 0$. For a freerider, the cost is $C \cdot (1-\delta_1)(1-\delta_2)(1-\delta_3)$ and drops to 0 if it is expelled from the system. The utility of a node out of the system is therefore 0 and necessarily $B > C$, otherwise nodes would gain nothing by joining the system. We can now express the expected utility u of a freerider as a function of its degree of freeriding Δ: If the freerider is detected (which occurs with probability $\alpha(\Delta)$), its utility drops to 0, otherwise (probability $1-\alpha(\Delta)$) its utility is the benefit of seeing the stream (i.e., B) minus the cost of (partially) uploading the stream, i.e., $C(1-\delta_1)(1-\delta_2)(1-\delta_3)$. That is:

$$\begin{aligned} u(\Delta) &= \alpha(\Delta)\times 0 + (1-\alpha(\Delta))\times(B - C(1-\delta_1)(1-\delta_2)(1-\delta_3)) \\ &= (1-\alpha(\Delta))\times(B - C(1-\delta_1)(1-\delta_2)(1-\delta_3)) \end{aligned} \quad (8)$$

As freeriders are utility-maximizing entity, they will choose the value of Δ that maximizes u.

In Figure 8, we also plot the freeriders' utility to determine their optimal strategy (i.e., their degree of freeriding) for different values of the benefit and of the cost (B/C). It can be observed that for $B/C = 1.1$ (i.e., the freeriders care slightly more about the stream than about their upload bandwidth) the utility is maximized for a degree of freeriding around $\delta = 0.025$, which corresponds to a gain (i.e., the fraction of upload bandwidth saved) of ~7%. For $B/C = 1.5$ however, the optimal strategy is to well-behave.

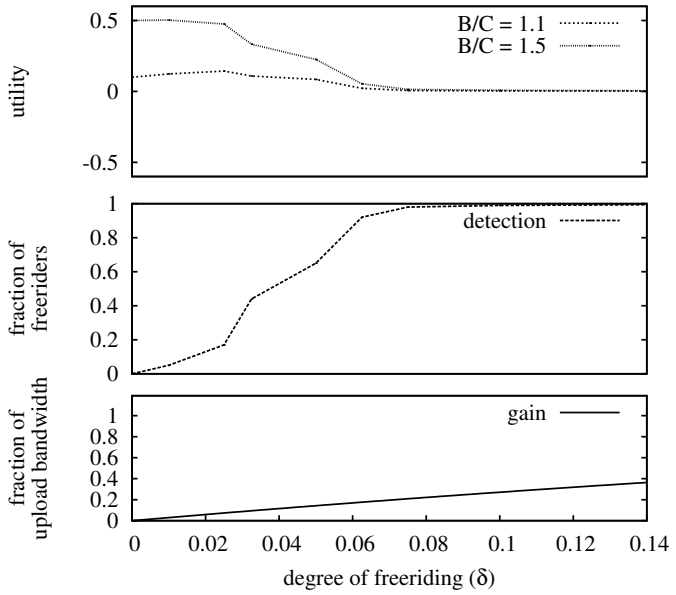

Figure 8: Proportion of freeriders detected by LiFTinG.

5.2. Entropy-based detection

For the sake of fairness and in order to prevent colluding nodes from covering each other up, LiFTinG includes an entropic check thus assessing the statistical validity of the partner selection. To this end, the entropy H of the distribution of the inspected node's former partners is compared to a threshold γ. The distribution of the entropy of honest nodes' histories depends on the peer sampling algorithm used and the random numbers generator. It can be estimated by simulations. Figure 9a depicts the distribution of entropy for a history of $n_h f = 600$ partners ($n_h = 50$ and $f = 12$) of a 10,000-node system using a full membership-based partner selection. The observed entropy ranges from 9.11 to 9.21 for a maximum reachable value of $\log_2(n_h f) = 9.23$. Similarly, the entropy of the fan-in multi-set $\mathcal{F}'_h$, i.e., nodes that selected the inspected node as partner, is depicted in Figure 9b. The observed entropy ranges from 8.98 to 9.34.

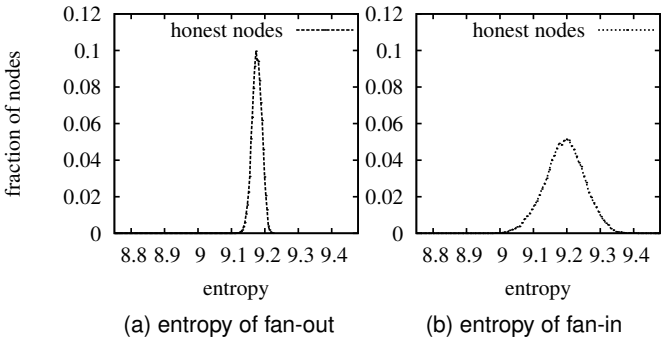

Figure 9: Entropy distribution (experimental pdf) of the nodes' histories using a full membership.

With $\gamma = 8.95$ the probability of wrongfully expelling a node during local auditing is negligible.

We now analytically determine to what extent a freerider can bias its partner selection without being detected by local auditing, given a threshold γ and a number of colluding nodes[6] m'. Consider a freerider that biases partner selection in order to favor colluding freeriders by choosing a freerider as partner with probability p_m and an honest node with probability $1 - p_m$. We seek the maximum value of p_m a freerider can use without being detected, function of γ and m'. Defining the probability law of the partner selection among honest nodes (resp. colluding nodes) by X (resp. by Y), the entropy of its fan-out is expressed as follows:

$$H(\mathcal{F}_h) = -p_m \log_2 p_m - (1-p_m)\log_2(1-p_m) + p_m H(X) + (1-p_m)H(Y) \ ,$$

as X and Y are independent. This quantity is maximized when X and Y are the uniform distribution. Therefore, to maximize the entropy of its history, a freerider must choose uniformly at random its partners in the chosen class, i.e., honest or colluding. In this case, the entropy of its history writes (for $m' < n_h f$):

$$H(\mathcal{F}_h) = -p_m \log_2\left(\frac{p_m}{m'}\right) - (1-p_m)\log_2\left(\frac{1}{n_h \cdot f}\right) \qquad (9)$$

Inverting numerically Formula (9), we deduce that for $\gamma = 8.95$ a freerider colluding with 25 other nodes can serve its colluding partners up to 15% of the time, without being detected.

In some cases, the random selection of nodes can be biased and therefore deviates from the uniform distribution. For instance, because inter-ISP traffic is expensive, ISPs sometime block peer-to-peer traffic between their own clients and clients of other ISPs [47, 50]. Also, nodes might prefer to communicate with close nodes (with respect to network distance) for improved latencies and performance. Such optimizations are considered as freeriding by LiFTinG and can make a node fail the statistical test, thus leading to its expulsion from the system. Provided that the size of the groups towards which the nodes bias the selection are large enough (typically much larger than the size of a coalition), LiFTinG can still distinguish between freeriders and honest nodes by relaxing the threshold of the entropic test, at the expense of an increased tolerance towards freeriders. This is achieved by calibrating the threshold as explained above with the typical size of a group (i.e., the average number of nodes that are located in the same country or that have the same ISP) instead of the system size n.

5.3. Communication Costs

In this section, we evaluate the overhead caused by LiFTinG on the content dissemination protocol. To this end, we compute the maximum number of verification and blame messages sent by a node during one gossip period. The overhead of the verifications is summarized in Table 4. Note that we do not consider statistical verifications in this section, as it does not imply a regular overhead but only sporadic message exchanges.

Direct verification. Direct verifications do not require any exchange of verification messages as they consist only in comparing the number of chunks requested by the verifier to the number of chunks it really received. However, direct verification can lead to the emission of f blames (to M managers). The communication overhead caused by direct verifications is therefore $O(M \cdot f)$ messages.

Direct cross-checking. In order to check that the chunks it sent during the previous gossip period are further proposed, the verifier polls the f partners of its f partners with probability p_{dcc}. Similarly, a node is polled by $p_{\text{dcc}} \cdot f^2$ nodes per gossip period on average and therefore sends $p_{\text{dcc}} \cdot f^2$ replies. Finally, a node sends the list of its current partners to the f nodes (on average) that served chunks to it in the last gossip period. In addition, as a node inspects its f partners, direct cross-checking can lead to the emission of a maximum of f blames (to M managers). The communication overhead caused by direct cross-checking is therefore $O(p_{\text{dcc}} \cdot f^2 + p_{\text{dcc}} \cdot M \cdot f)$ messages. Setting p_{dcc} to $1/f$ the overhead is $O(M + f)$.

[6] Note that, as the length of a history is relatively small, we do not take into account the number of nodes that join the system during this time interval.

direct verifications (messages)	0
direct verifications (blames)	$O(M \cdot f)$ for the verifier
direct cross-check (messages)	$O(p_{\text{dcc}} f^2)$ for the verifier $O(p_{\text{dcc}} f)$ for the inspected node $O(p_{\text{dcc}} f^2)$ for the each witness
direct cross-check (blames)	$O(p_{\text{dcc}} \cdot M \cdot f)$ for the verifier

Table 4: Overhead of verifications.

The number of messages sent by LiFTinG is $O(M \cdot f)$. This has to be compared to the number of messages sent by the three-phase gossip protocol itself, specifically $f(2 + |\mathcal{R}|)$ – where $\mathcal{R}$ is the set of requested chunks, the two additional messages are the proposal and the request. The overhead of LiFTinG is even more negligible when taking into account the size of the chunks sent by a node, which is several orders of magnitude larger than the verification and blame messages. Finally, as $f \sim \ln(n)$, both the three-phase protocol and LiFTinG scale with the number of nodes. Finally, note that setting f to $\ln(n)$ in an infect-and-die gossip protocol is sufficient to ensure that the content is successfully broadcast with high probability [31]. Thus, both the three-phase protocol and LiFTinG scale with the number of nodes.

A posteriori verification and cross-checking. Assume that nodes verify the nodes they manage. Every period of time, a node picks one such node and triggers a verification. The cost of obtaining/providing the inspected node history is $O(n_h f)$ (for the inspected node and for the verifier). For each entry in the history, the verifier asks for a confirmation to the corresponding node (with a given probability p_c). This leads to a cost of $O(p_c n_h f)$ (for the verifier and for the polled nodes as they answer to $p_c n_h f$ such confirmation requests on average, each with a unit cost).

6. Evaluation and Experimental Results

We now evaluate LiFTinG on top of the gossip-based streaming protocol described in [14], over the PlanetLab testbed. We describe the experimental setup in Section 6.1. We evaluate the performance of LiFTinG showing its small overhead in Section 6.2 and its precision and speed at detecting freeriders in Section 6.3.

6.1. Experimental Setup

We deploy and execute LiFTinG on a 300 PlanetLab node testbed, broadcasting a stream of 674 kbps in the presence of 10% of freeriders. The freeriders *(i)* contact only $\hat{f} = 6$ random partners ($\delta_1 = 1/7$), *(ii)* propose only 90% of what they receive ($\delta_2 = 0.1$) and finally *(iii)* serve only 90% of what they are requested ($\delta_3 = 0.1$). The fan-out of all nodes is set to 7 and the gossip period is set to 500 ms. The blaming architecture uses $M = 25$ managers for each node.

6.2. Practical Cost

We report on the overhead measurements of direct and *a posteriori* verifications (including blame messages sent to the managers) for different stream rates.

Direct verifications. Table 5 gives the bandwidth overhead of the direct verifications of LiFTinG for three values of p_{cc}. Note that the overhead is not null when $p_{\text{cc}} = 0$ because acknowledgment messages are always sent. Yet, we observe that the overhead is negligible when $p_{\text{cc}} = 0$ (i.e., when the system is healthy) and remains reasonable when $p_{\text{cc}} = 1$ (i.e., when the system needs to be purged from freeriders).

	direct			**a posteriori**
	$p_{cc} = 0$	$p_{cc} = 0.5$	$p_{cc} = 1$	
674 kbps	1.07%	4.53%	8.01%	3.60%
1082 kbps	0.69%	3.51%	5.04%	2.89%
2036 kbps	0.38%	2.80%	2.76%	1.74%

Table 5: Practical bandwidth overhead.

A posteriori verifications. A history message contains n_h entries. Each entry consists of f nodes identifiers and the chunk IDs that were proposed. Both the fan-out and fan-in histories are sent upon a posteriori verification.

Besides the entropic checks, *a posteriori* cross-checking is performed on a subset of the fan-out or fan-in entries. We measure the maximum overhead, that is when the whole fan-out and fan-in histories are cross-checked. The overhead incurred by *a posteriori* verifications in our experimental setup (i.e., a history size $n_h = 50$, a gossip period of 500 ms, a fan-out of $f = 7$ and *a posteriori* verification period of $h = 25$ s) is given in Table 5.

6.3. Experimental Results

We executed LiFTinG with $p_{\text{cc}} = 1$ and $p_{\text{cc}} = 0.5$. Figure 10 depicts the scores obtained after 25, 30 and 35 seconds when running direct verifications and cross-checking. The scores are compensated as explained in the analysis, assuming a loss rate of 4% (average value for UDP packets observed on PlanetLab).

The two cumulative distribution functions, for honest nodes and freeriders, are clearly separated. The threshold for expelling freeriders is set to -9.75 (as specified in the analysis). In Figure 10b ($p_{\text{cc}} = 1$, after 30 s) the detection mechanism expels 86% of the freeriders and 12% of the honest nodes. In other words, after 30 seconds, 14% of freeriders are not yet detected and 12% represent false positives that mainly correspond to honest nodes that suffer from very poor connection (e.g., limited connectivity, message losses and bandwidth limitation). These nodes do not deliberately freeride, but their connection does not allow them to contribute their fair share. This is acceptable as such nodes should not have been allowed to join the system in the first place. As expected, with p_{cc} set to 0.5 the detection is slower but not twice as slow. Effectively, with nodes freeriding with $\delta_3 > 0$ (i.e., partial serves) the direct checking blames freeriders without the need for any cross-checking. This explains why the detection after only 35 seconds with $p_{\text{cc}} = 0.5$ (Figure 10f) is comparable with the detection after 30 seconds with $p_{\text{cc}} = 1$ (Figure 10b).

Due to the dynamic nature of live streaming, being able to expel freeriders after less than 1 minute allows us to drastically reduce their viewing experience. Indeed, because of LiFTinG's exclusion mechanism, a freerider will experience frequent pauses, i.e., every time it is excluded from the system,

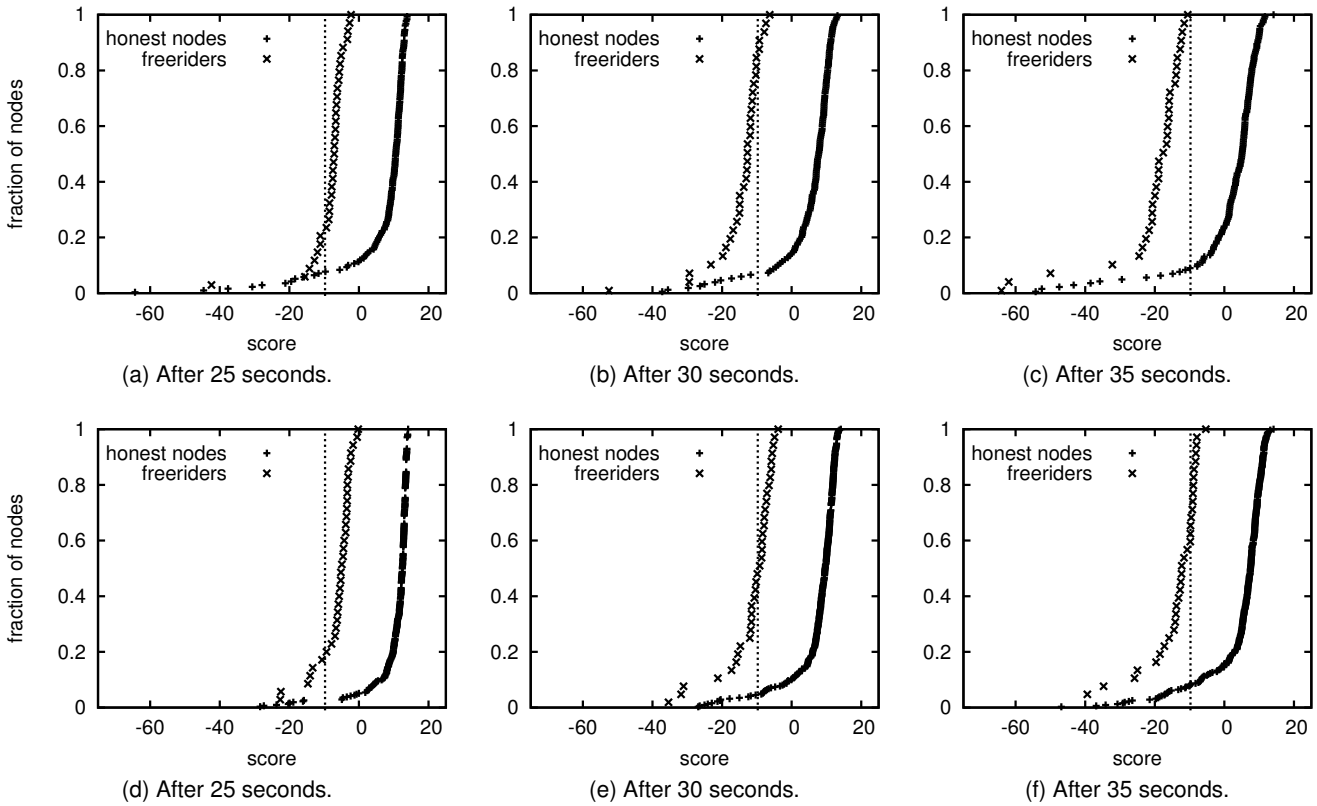

Figure 10: Scores CDF for honest nodes and freeriders, with $p_{cc} = 1$ (top row) and $p_{cc} = 0.5$ (bottom row).

during the viewing. The duration of such pauses is increased by the fact that the freerider needs to change its identifier (e.g., its IP) to be able to rejoin the system and by the bootstrapping time inherent to the dissemination protocol (e.g., buffering time, delay before the node's ID is available to the node selection). Hence, running a posteriory verification every few minutes on average is enough to significantly degrade the freeriders's experience. For the same reasons, *whitewashing* (i.e., leaving and joining the system with a fresh identifier to reset its score) is not a viable solution for freeriders.

As stated in the analysis, we observe that the gap between the two cumulative distribution functions widens over time. However, the variance of the score does not decrease (for both honest nodes and freeriders). This is because, in the analysis, we considered that the blames applied to a given node during distinct gossip periods were independent and identically distributed (i.i.d.). In practice however, successive gossip periods are correlated. Indeed, a node with a poor connection is usually blamed more than nodes with high capabilities, and this remains true over the whole experiment.

7. Related Work

TfT distributed incentives have been broadly used to deal with freeriders in file sharing systems based on symmetric exchanges, such as BitTorrent [10]. However, there are a number of attacks, mainly targeting the opportunistic unchoking mechanism (i.e., asymmetric push), allowing freeriders to download contents with no or a very small contribution [38, 45, 51]. Many other incentive schemes have been proposed, in particular for mesh overlays (e.g., reputation-based [29, 41, 42, 49]) or trees (e.g., [57], market-based [44], payment-based [53]) systems. However, most of them either rely on a central authority to maintain reputation or consider static overlays for easier auditing and real-time verifications. In [46], the authors propose OneHop reputations, a system in which peers rely on the peers they interacted with in the past to assess the reputation of the peers they could interact with: if a peer p never interacted with p' in the past, it looks for a peer p'' that interacted with it (i.e., p) and with p' in the past, and asks p'' about the reputation of p'. However, while such a peer can be found most of the times in small swarms in which peers collaborate with many other peers and keep track of all their previous interactions, this might not be the case for large-scale epidemic systems with hard scalability constraints. In [28, 29], the author propose EigenTrust, a distributed algorithm for reputation management in peer-to-peer networks, based on the number of satisfactory/unsatisfactory pairwise interactions. The authors also propose a (secure) distributed version of EigenTrust that relies, similarly to LiFTinG, on the peers to act as score managers (for each other) for aggregating, computing and maintaining the reputation of other peers. The focus and the contribution of EigenTrust are dual to those of LiFTinG: EigenTrust relies on basic input from the peers (i.e., generic binary input that represents whether an interaction was satisfactory) and proposes a complex aggregation and management scheme for reputation

scores; LiFTinG however, relies on fine-grained input from the peers (i.e., blames, tightly related to the dissemination protocol, with integer values that reflect the seriousness of the deviation– which is one of the main contributions) and makes use of a simple aggregation scheme (i.e., sum of the blame values). Note also that EigenTrust can raise scalability issues in the context of epidemic dissemination as it involves computations on the scores given by all the peers that interacted with a given peer.

FlightPath (built on top of BAR Gossip) [36] is a gossip-based streaming application that fights against freeriding by using verifications on partner selection and chunk exchanges. FlightPath operates in a gossip fashion for partner selection and is composed of opportunistic pushes performed by altruistic nodes (essential for the efficiency of the protocol) and balanced pairwise exchanges secured by TfT. The randomness of partner selection is verified by means of a pseudo-random number generator with signed seeds, and symmetric exchanges are made robust by using cryptographic primitives. FlightPath prevents attacks on opportunistic pushes by turning them into symmetric exchanges: each peer must reciprocate with junk chunks when opportunistically unchoked. This results in a non-negligible waste of bandwidth. It is further demonstrated in [22] that BAR Gossip presents scalability issues, not to mention the overhead of cryptography.

PeerReview [20] deals with malicious nodes following an accountability approach. Peers maintain signed logs of their actions that can be checked by using a reference implementation running in addition to the application. When combined with CSAR [4], PeerReview can be applied to non-deterministic protocols. However, the intensive use of cryptography and the sizes of the logs maintained and exchanged drastically reduce the scalability of this solution. In addition, the validity of PeerReview relies on the fact that messages are always received, which is not the case over the Internet.

The case of malicious participants is considered in the context of generic gossip protocols, i.e., consisting of state exchanges and updates [26]. This system relies on cryptography for signing messages between peers and does not consider malicious behaviors that stem from the partner selection, i.e., biasing the random choices. In addition, they do not tackle the problem of collusion.

The two approaches that relate the most to LiFTinG are the distributed auditing protocol proposed in [22] and the passive monitoring protocol proposed in [30]. In the first protocol, freeriders are detected by cross-checking their neighbors' reports. The latter focuses on gossip-based search in the Gnutella network. The peers monitor the way their neighbors forward/answer queries in order to detect freeriders and query droppers. In [8], the authors propose a protocol to detect and exclude freeriders in the CAN peer-to-peer data structure by monitoring how peers forward and answer routing messages. Yet, contrarily to LiFTinG– which is based on random peer selection – in both protocols the peers's neighborhoods are static, forming a fixed mesh overlay. These techniques thus cannot be applied to gossip protocols. In addition, the situation where colluding peers cover each other up (not addressed in the papers) makes such monitoring protocols vain.

In [2], the authors study the impact of measures against the peers, such as the exclusion of peers, on several metrics, including the extinction time of the shared file, in epidemic dissemination systems. The results of this work can be used to evaluate the efficacy of LiFTinG in terms of its effect on the global performance of the system.

8. Conclusion

We have presented LiFTinG, a protocol for tracking freeriders in gossip-based asymmetric data dissemination systems. Beyond the fact that LiFTinG deals with the inherent randomness of the protocol, LiFTinG precisely relies on this randomness to robustify, with very low overhead, its verification mechanisms against colluding freeriders . We provided a theoretical analysis of LiFTinG that enables the system designers to set its parameters to their optimal values and characterizes its performance backed up by extensive simulations. We reported on our PlanetLab experimentation, that demonstrates the practicability and efficiency of LiFTinG.

We believe that, beyond gossip protocols, LiFTinG can be used to secure the asymmetric component of TfT-based protocols, namely *opportunistic unchoking*, which is considered to constitute their Achilles' heel [38, 51]. We can also envision a scheme in which peers are rewarded (by increasing their scores) when they altruistically push chunks to other peers, even after completing the download (i.e., seeding, for P2P file download systems). As future work, we intend to model a content-dissemination system secured by LiFTinG as a game and study the strategies of freeriders and the possible equilibria.

References

References

[1] E. Adar, B. Huberman, Free Riding on Gnutella, First Monday 5 (2000).

[2] E. Altman, P. Nain, A. Shwartz, Y. Xu, Predicting the Impact of Measures Against P2P Networks on the Transient Behaviors, in: INFOCOM'11: Proc. of the 30th IEEE Conference on Computer Communications, pp. 1440–1448. doi:10.1109/INFCOM.2011.5934931.

[3] F. Azzedin, Trust-based taxonomy for free riders in distributed multimedia systems, in: HPCS'10: Proc. of the 2010 International Conference on High Performance Computing and Simulation, pp. 362–369. doi:10.1109/HPCS.2010.5547108.

[4] M. Backes, P. Druschel, A. Haeberlen, D. Unruh, CSAR: A Practical and Provable Technique to Make Randomized Systems Accountable, in: NDSS'09: Proc. of the 16th Annual Network & Distributed System Security Symposium, pp. 341–353.

[5] T. Bonald, L. Massoulié, F. Mathieu, D. Perino, A. Twigg, Epidemic Live Streaming: Optimal Performance Trade-offs, in: SIGMETRICS'08: Proc. of the 2008 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, pp. 325–336. doi:10.1145/1375457.1375494.

[6] E. Bortnikov, M. Gurevich, I. Keidar, G. Kliot, A. Shraer, Brahms: Byzantine Resilient Random Membership Sampling, Computer Networks 53 (2009) 2340–2359. doi:10.1145/1400751.1400772.

[7] S. Buchegger, J.Y. Le Boudec, Coping with False Accusations in Misbehavior Reputation Systems for Mobile Ad-hoc Networks, Technical Report, EPFL, 2003. URL: http://infoscience.epfl.ch/record/467.

[8] E. Buchmann, K. Böhm, FairNet: How to Counter Free Riding in Peer-to-Peer Data Structures, in: CoopIS'04: Proc. of the 2004 International Conference on Cooperative Information Systems, pp. 337–354. doi:10.1007/978-3-540-30468-5_22.

[9] K.H. Chan, S.H. Chan, A. Begen, SPANC: Optimizing Scheduling Delay for Peer-to-Peer Live Streaming, IEEE Transactions on Multimedia 12 (2010) 743–753. doi:10.1109/TMM.2010.2053524.

[10] B. Cohen, Incentives Build Robustness in BitTorrent, in: P2PEcon'03: Proc. of the 1st Workshop on Economics of Peer-to-Peer Systems, pp. 1–5.

[11] T. Cover, J. Thomas, Elements of Information Theory, John Wiley and Sons, Inc., 1991.

[12] M. Deshpande, B. Xing, I. Lazardis, B. Hore, N. Venkatasubramanian, S. Mehrotra, CREW: A Gossip-based Flash-Dissemination System, in: ICDCS'06: Proc of the 26th IEEE International Conference on Distributed Computing Systems, pp. 45–45. doi:10.1109/ICDCS.2006.24.

[13] P.T. Eugster, R. Guerraoui, S.B. Handurukande, P. Kouznetsov, A.M. Kermarrec, Lightweight Probabilistic Broadcast, ACM Transactions on Computer Systems 21 (2003) 341–374. doi:10.1145/945506.945507.

[14] D. Frey, R. Guerraoui, A.M. Kermarrec, M. Monod, V. Quéma, Stretching Gossip with Live Streaming, in: DSN'09: Proc. of the 2009 IEEE/IFIP International Conference on Dependable Systems and Networks, pp. 259–264. doi:10.1109/DSN.2009.5270330.

[15] D. Frey, R. Guerraoui, B. Koldehofe, A.M. Kermarrec, M. Mogensen, M. Monod, V. Quéma, Heterogeneous Gossiping, in: Middleware'09: Proc. of the ACM/IFIP/USENIX 10th International Middleware Conference, pp. 42–61. doi:10.1007/978-3-642-10445-9_3.

[16] A. Ganesh, A.M. Kermarrec, L. Massoulié, SCAMP: Peer-to-peer Lightweight Membership Service for Large-scale Group Communication, in: NGC'01: Proc. of the 3rd International Workshop on Networked Group Communication, pp. 44–55. doi:10.1007/3-540-45546-9_4.

[17] J. Gerard, H. Cai, J. Wang, Alliatrust: A Trustable Reputation Management Scheme for Unstructured P2P Systems, in: GPC'06: Proc. of the 1st International Conference on Advances in Grid and Pervasive Computing, pp. 115–125. doi:10.1007/11745693_12.

[18] C. Gkantsidis, P. Rodriguez, Network Coding for Large-Scale Content Distribution, in: INFOCOM'05: Proc. of the 24th IEEE Conference on Computer Communications, pp. 2235–2245. doi:10.1109/INFCOM.2005.1498511.

[19] R. Guerraoui, K. Huguenin, A.M. Kermarrec, M. Monod, S. Prusty, LiFTinG: Lightweight Freerider-Tracking Protocol in Gossip, in: Middleware'10: Proc. of the ACM/IFIP/USENIX 11th International Middleware Conference, pp. 313–333. doi:10.1007/978-3-642-16955-7_16.

[20] A. Haeberlen, P. Kouznetsov, P. Druschel, PeerReview: Practical Accountability for Distributed Systems, in: SOSP'07: Proc. of 21st ACM SIGOPS Symposium on Operating Systems Principles, pp. 175–188. doi:10.1145/1323293.1294279.

[21] G. Hardin, The Tragedy of the Commons, Science 162 (1968) 1243–1248. doi:10.1126/science.162.3859.1243.

[22] M. Haridasan, I. Jansch-Porto, R. Van Renesse, Enforcing Fairness in a Live-Streaming System, in: MMCN'08: Proc. of the 2008 Conference on Multimedia Computing and Networking, pp. 1–13. doi:10.1117/12.775127.

[23] T. Ho, M. Mdard, R. Koetter, D. Karger, M. Effros, J. Shi, B. Leong, A Random Linear Network Coding Approach to Multicast, IEEE Transactions on Information Theory 52 (2006) 4413–4430. doi:10.1109/TIT.2006.881746.

[24] K. Hoffman, D. Zage, C. Nita-Rotaru, A Survey of Attack and Defense Techniques for Reputation Systems, ACM Computing Surveys 42 (2009) 1–31. doi:10.1145/1592451.1592452.

[25] D. Hughes, G. Coulson, J. Walkerdine, Free Riding on Gnutella Revisited: The Bell Tolls?, IEEE Distributed Systems Online 6 (2005) 1–18. doi:10.1109/MDSO.2005.31.

[26] M. Jelasity, A. Montresor, O. Babaoglu, Detection and Removal of Malicious Peers in Gossip-Based Protocols, in: FuDiCo'04: Proc. of the 2nd Workshop on Future Directions in Distributed Computing, pp. 1–4.

[27] M. Jelasity, S. Voulgaris, R. Guerraoui, A.M. Kermarrec, M. van Steen, Gossip-based Peer Sampling, ACM Transactions on Computer Systems 25 (2007) 1–36. doi:10.1145/1275517.1275520.

[28] S.D. Kamvar, M.T. Schlosser, H. Garcia-Molina, Incentives for Combatting Freeriding on P2P Networks, in: Euro-Par'03: Proc. of the 9th International Conference on Parallel and Distributed Computing, pp. 1273–1279. doi:10.1007/978-3-540-45209-6_171.

[29] S.D. Kamvar, M.T. Schlosser, H. Garcia-Molina, The Eigentrust Algorithm for Reputation Management in P2P Networks, in: WWW'03: Proc. of the 12th International Conference on the World Wide Web, pp. 640–651. doi:10.1145/775152.775242.

[30] M. Karakaya, I. Körpeoğlu, O. Ulusoy, Counteracting Free-riding in Peer-to-Peer Networks, Computer Networks 52 (2008) 675–694. doi:10.1016/j.comnet.2007.11.002.

[31] A.M. Kermarrec, L. Massoulié, A. Ganesh, Probabilistic Reliable Dissemination in Large-Scale Systems, IEEE Transactions on Parallel and Distributed Systems 14 (2003) 248–258. doi:10.1109/TPDS.2003.1189583.

[32] A.M. Kermarrec, A. Pace, V. Quéma, V. Schiavoni, NAT-Resilient Gossip Peer Sampling, in: ICDCS'09: Proc. of the 29th IEEE International Conference on Distributed Computing Systems, pp. 360–367. doi:10.1109/ICDCS.2009.44.

[33] V. King, J. Saia, Choosing a Random Peer, in: PODC'04: Proc. of the 23rd Annual ACM Symposium on Principles of Distributed Computing, pp. 125–130. doi:10.1145/1011767.1011786.

[34] R. Krishnan, M. Smith, Z. Tang, R. Telang, The Impact of Free-Riding on Peer-to-Peer Networks, in: HICSS'04: Proc. of the 37th Annual Hawaii International Conference on System Sciences, pp. 1–10. doi:10.1109/HICSS.2004.1265472.

[35] B. Li, Y. Qu, Y. Keung, S. Xie, C. Lin, J. Liu, X. Zhang, Inside the New Coolstreaming: Principles, Measurements and Performance Implications, in: INFOCOM'08: Proc. of the 27th IEEE Conference on Computer Communications, pp. 1031–1039. doi:10.1109/INFOCOM.2008.157.

[36] H. Li, A. Clement, M. Marchetti, M. Kapritsos, L. Robinson, L. Alvisi, M. Dahlin, FlightPath: Obedience vs Choice in Cooperative Services, in: OSDI'08: Proc. of the 8th USENIX Conference on Operating Systems Design and Implementation, pp. 355–368.

[37] Q. Lian, Z. Zhang, M. Yang, B.Y. Zhao, Y. Dai, X. Li, An Empirical Study of Collusion Behavior in the Maze P2P File-Sharing System, in: ICDCS'07: Proc of the 27th IEEE International Conference on Distributed Computing Systems, p. 56. doi:10.1109/ICDCS.2007.84.

[38] T. Locher, P. Moor, S. Schmid, R. Wattenhofer, Free Riding in BitTorrent is Cheap, in: HotNets-V: Proc. of the 5th Workshop on Hot Topics in Networks (2006), pp. 85–90.

[39] N. Magharei, R. Rejaie, PRIME: Peer-to-Peer Receiver-Driven Mesh-Based Streaming, IEEE/ACM Transactions on Networking 17 (2009) 1052–1065. doi:10.1109/TNET.2008.2007434.

[40] S. Marti, H. Garcia-Molina, Taxonomy of Trust: Categorizing P2P Reputation Systems, Computer Networks 50 (2006) 472 – 484. doi:j.comnet.2005.07.011.

[41] M. Meulpolder, J. Pouwelse, D. Epema, H. Sips, BarterCast: A Practical Approach to Prevent Lazy Freeriding in P2P Networks, in: IPDPS'09: Proc. of the IEEE International Symposium on Parallel & Distributed Processing, pp. 1–8. doi:10.1109/IPDPS.2009.5160954.

[42] A. Montazeri, B. Akbari, Mesh-Based P2P Video Streaming with a Distributed Incentive Mechanism, in: ICOIN'11: Proc. of the 2011 International Conference on Information Networking, pp. 108–113. doi:10.1109/ICOIN.2011.5723143.

[43] R. Morales, I. Gupta, AVMON: Optimal and Scalable Discovery of Consistent Availability Monitoring Overlays for Distributed Systems, IEEE Transactions on Parallel and Distributed Systems 20 (2009) 446–459. doi:10.1109/TPDS.2008.84.

[44] A. Payberah, F. Rahimian, S. Haridi, J. Dowling, Sepidar: Incentivized Market-Based P2P Live-Streaming on the Gradient Overlay Network, in: ISM'10: Proc. of the 2010 IEEE International Symposium on Multimedia, pp. 1–8. doi:10.1109/ISM.2010.11.

[45] M. Piatek, T. Isdal, T. Anderson, A. Krishnamurthy, A. Venkataramani, Do Incentives Build Robustness in BitTorrent?, in: NSDI'07: Proc. of the 4th USENIX Symposium on Networked Systems Design and Implementation, pp. 1–14.

[46] M. Piatek, T. Isdal, A. Krishnamurthy, T. Anderson, One Hop Reputations for Peer to Peer File Sharing Workloads, in: NSDI'08: Proc. of the 5th USENIX Symposium on Networked Systems Design and Implementation, pp. 1–14.

[47] F. Picconi, L. Massoulie, ISP Friend or Foe? Making P2P Live Streaming ISP-Aware, in: ICDCS'09: Proc of the 29th IEEE International Con-

ference on Distributed Computing Systems, pp. 413–422. doi:10.1109/ICDCS.2009.37.

[48] J. Rosenberg, R. Mahy, P. Matthews, D. Wing, Session Traversal Utilities for NATs (STUN), Technical Report RFC 5389, IETF, 2008.

[49] A. Satsiou, L. Tassiulas, Reputation-Based Resource Allocation in P2P Systems of Rational Users, IEEE Transactions on Parallel and Distributed Systems 21 (2010) 466–479. doi:10.1109/TPDS.2009.80.

[50] Z. Shen, R. Zimmermann, ISP-Friendly P2P Live Streaming: A Roadmap to Realization, ACM Transactions on Multimedia Computing, Communications and Applications 8 (2012) 11:1–11:20. doi:10.1145/2089085.2089088.

[51] M. Sirivianos, J. Park, R. Chen, X. Yang, Free-riding in BitTorrent with the Large View Exploit, in: IPTPS'07: Proc. of the 6th International Workshop on Peer-to-Peer Systems, pp. 1–6.

[52] M.A. Stephens, EDF Statistics for Goodness of Fit and Some Comparisons, Journal of the American Statistical Association 69 (1974) 730–737.

[53] G. Tan, S.A. Jarvis, A Payment-Based Incentive and Service Differentiation Scheme for Peer-to-Peer Streaming Broadcast, IEEE Transactions on Parallel and Distributed Systems 19 (2008) 940–953. doi:10.1109/TPDS.2007.70778.

[54] D.N. Tran, B. Min, J. Li, L. Subramanian, Sybil-Resilient Online Content Voting, in: NSDI'09: Proc. of the 6th USENIX Symposium on Networked Systems Design and Implementation, pp. 15–28.

[55] V. Venkataraman, K. Yoshida, P. Francis, Chunkyspread: Heterogeneous Unstructured Tree-Based Peer-to-Peer Multicast, in: ICNP'06: Proc. of the 14th IEEE International Conference on Network Protocols, pp. 2–11. doi:10.1109/ICNP.2006.320193.

[56] M. Wang, B. Li, R2: Random Push with Random Network Coding in Live Peer-to-Peer Streaming, IEEE Journal on Selected Areas in Communications 25 (2007) 1655–1666. doi:10.1109/JSAC.2007.071205.

[57] S. Yang, X. Wang, An Incentive Mechanism for Tree-based Live Media Streaming Service, Journal of Networks 5 (2010) 57–64.

[58] N. Zeilemaker, M. Capotă, A. Bakker, J. Pouwelse, Tribler: P2P Media Search and Sharing, in: MM'11: Proc. of the 19th ACM International Conference on Multimedia, pp. 739–742. doi:10.1145/2072298.2072433.

[59] M. Zhang, Q. Zhang, L. Sun, S. Yang, Understanding the Power of Pull-Based Streaming Protocol: Can We Do Better?, IEEE Journal on Selected Areas in Communications 25 (2007) 1678–1694. doi:10.1109/JSAC.2007.071207.

[60] X. Zhang, H. Hassanein, A Survey of Peer-to-Peer Live Video Streaming Schemes - An Algorithmic Perspective, Computer Networks 56 (2012) 3548–3579. doi:10.1016/j.comnet.2012.06.013.